Postel

7139

Vint Cerf - UCLA

Eric Harslem - Rand

John Heafner - Rand

Bob Metcalfe - MIT

Jim White - UCSB

RFC 194
NIC 7139
Category: D.4
Updates: None
Obsoletes: None

Received 2 Aug 71

THE DATA RECONFIGURATION SERVICE --

COMPILER/INTERPRETER IMPLEMENTATION NOTES

I. NEW FEATURES OF THE LANGUAGE

1. The meaning of S(#,E,,1) is only find an arbitrary number (≤256) of EBCDIC characters and store them in identifier S. This descriptor is terminated only by an invalid EBCDIC or by exceeding maximum permissible character count (256).

2. The assignment (S ·<=· T) causes all attributes of identifier T to be given to S, i.e., length, type, and contents.

3. (S ·<=· T || X) concatenates X onto the right-hand side of T and stores the result in S. If T and X are binary the resulting value has a length equal to the sum L(T) + L(X).

4. T(X) joins L(X) and V(X) as a built-in identifier function.

   T(X) = type of identifier X.

   L(X) = length of contents of X.

   V(X) = contents of X converted to binary (decimal → binary is presently the only transformation).

5. New types ED and AD are EBCDIC and ASCII encoded decimal, respectively. These have been added to complement the V(X) function.

6. New type SB has been added as signed binary. Type B is a logical binary string.

7. The syntactic notation for return-from-a-form has been changed. See new syntax.

II. NEW SYNTAX

```
form              :: = rule | form
rule              :: = label  inputstream outputstream;
label             :: = INTEGER | null
inputstream       :: = terms | null
terms             :: = term | terms, term
outputstream      :: = :terms | null
term              :: = identifier | identifier descriptor |
                       descriptor | comparator
identifier        :: = <alpha followed by 0-3 alphamerics>
descriptor        :: = (replicationexpr,  datatype, valueexpr,
                       lengthexpr control)
comparator        :: = (concatexpr connective concatexpr control) |
                       (identifier .<=. concatexpr control)
replicationexpr   :: = # | arithmetricexpr | null
datatype          :: = B | O | X | E | A | ED | AD | SB | T (identifier)
valueexpr         :: = concatexpr | null
lengthexpr        :: = arithmeticexpr | null
connective        :: = .LE. | .LT. | .GT. | .GE. | .EQ. | .NE.
concatexpr        :: = value | concatexpr    value
value             :: = literal | arithmeticexpr
arithmeticexpr    :: = primary | arithmeticexpr operator primary
primary           :: = identifier | L(identifier) | V(identifier) |
                         INTEGER
operator          :: = + | - | * | /
literal           :: = literaltype "string"

literaltype       :: = B | Ø | X | E | A | ED | AD | SB
string            :: = <from 0 to 256 chars>
control           :: = :options |. null
options           :: = SFUR (arithmeticexpr)   |  SFUR (arithmeticexpr),
                       SFUR (arithmeticexpr)
SFUR              :: = S | F | U | SR | FR | UR
```

III. THE FORM INTERPRETER

Interpreter Overview

The interpreter is a simple minded machine having the virtue of helping the compiler writer by providing a rather powerful instruction set for hard-to-compile operations. Figure 1 shows the machine configuration:

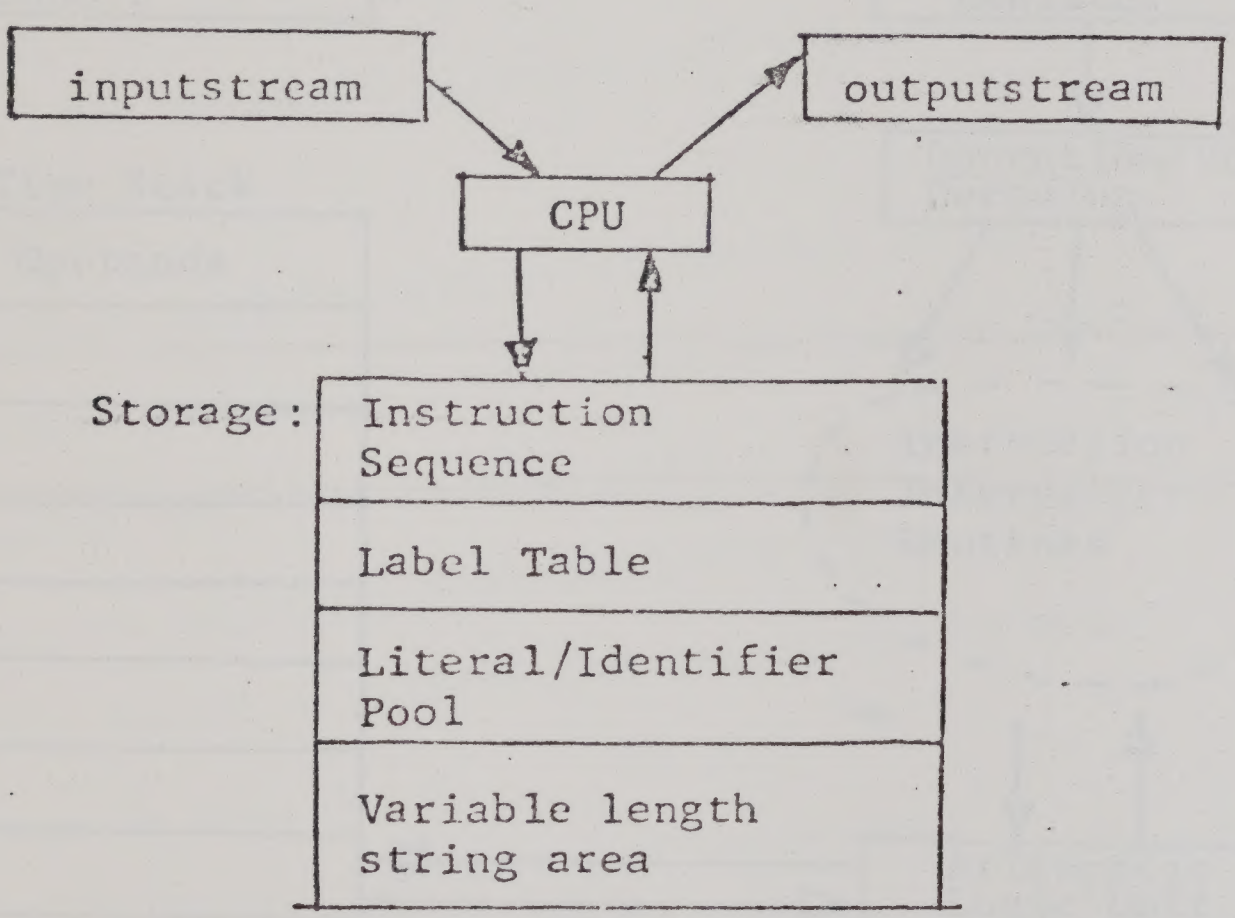


Fig. 1. Form Interpreter

The CPU is a box full of miscellaneous parts, the most important being the Arithmetic Logic Unit and the instruction decoding unit. The CPU also maintains a collection of state registers to keep track of what it is doing. Figure 2 shows the rough layout.

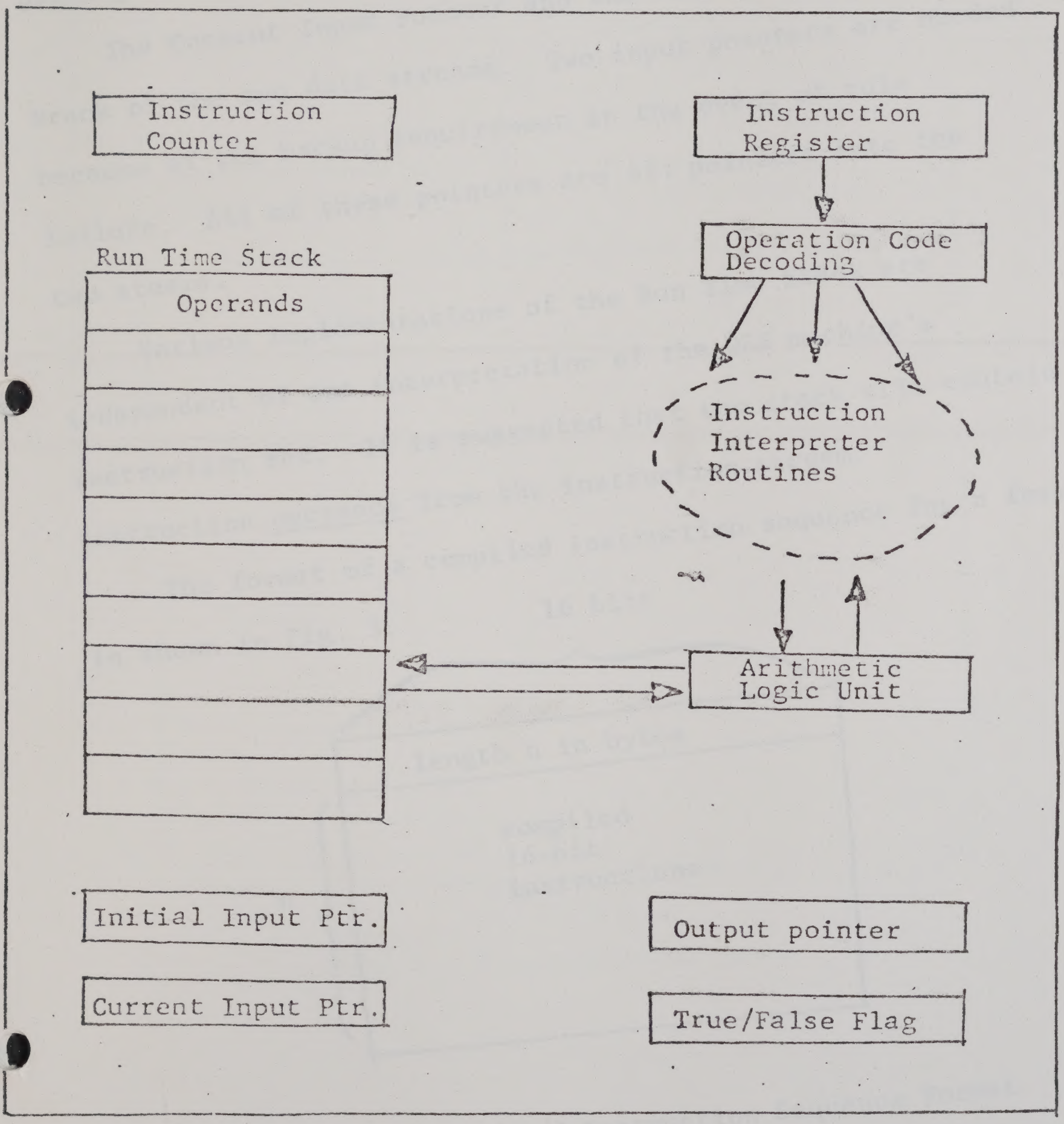
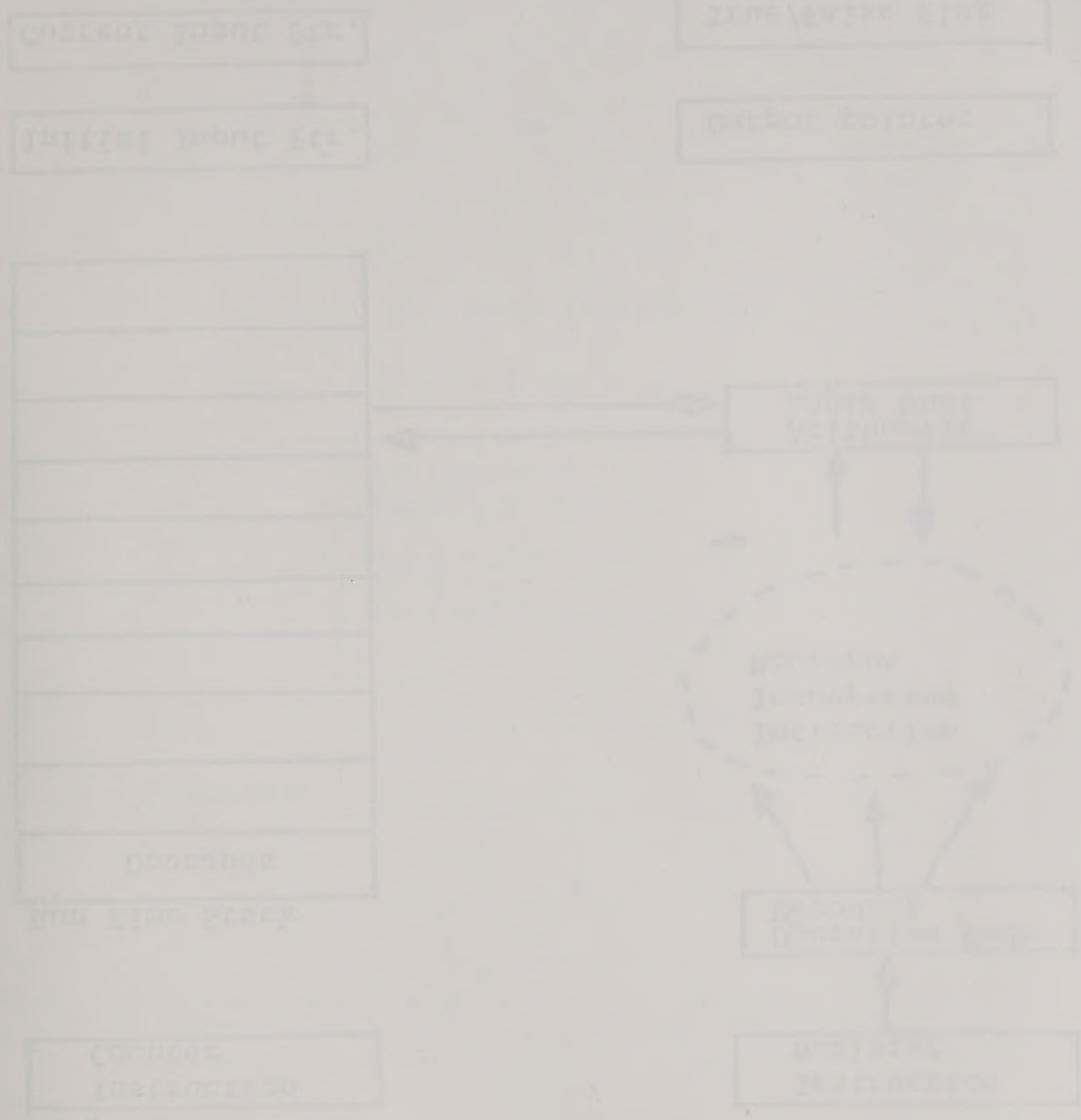


Fig. 2. The Central Processor



The CPU is a stack machine driven by a Polish postfix instruction sequence. Operands placed on the Run Time Stack are used for arithmetic expression evaluation and for parameter passing between the interpreter and the built-in functions.

The Current Input Pointer and the Output Pointer keep track of the two data streams. Two input pointers are needed because of the backup requirement in the event of rule failure. All of these pointers are *bit* pointers into the two steams.

Various implementations of the Run Time Stack are independent of the interpretation of the DRS machine's instruction set. It is suggested that the stack will contain instruction operands from the instruction stream.

The format of a compiled instruction sequence for a form is shown in Fig. 3.

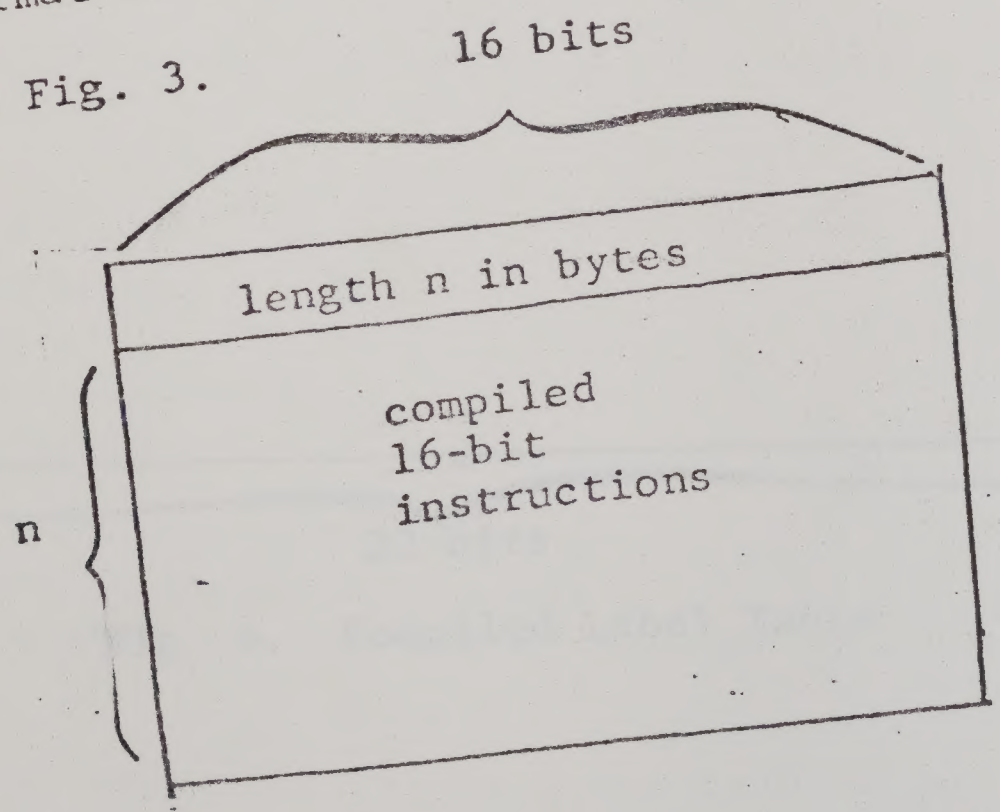


Fig. 3. Compiled Instruction Sequence Format

The format of the compiled Label Table is shown in Fig. 4.

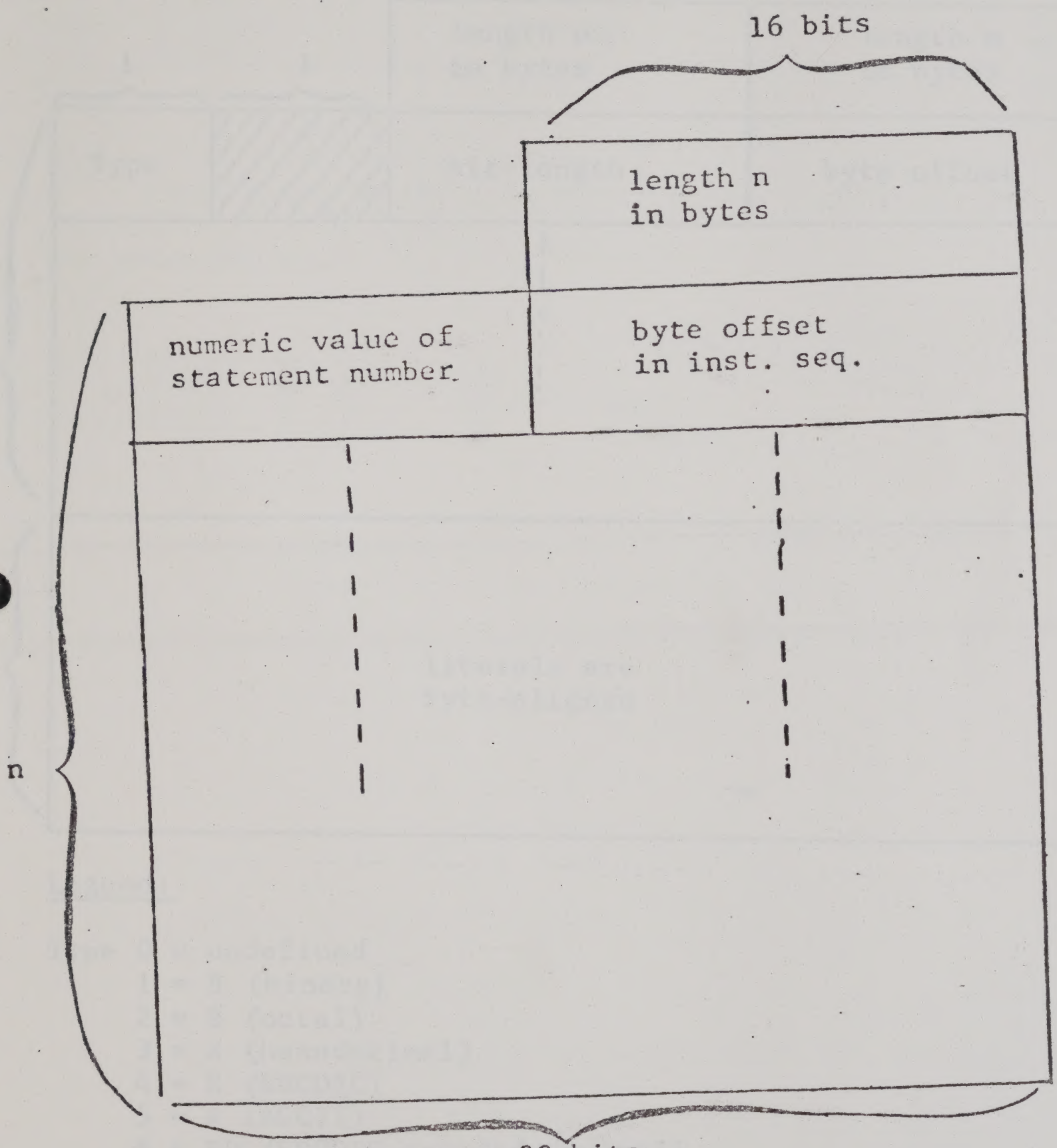
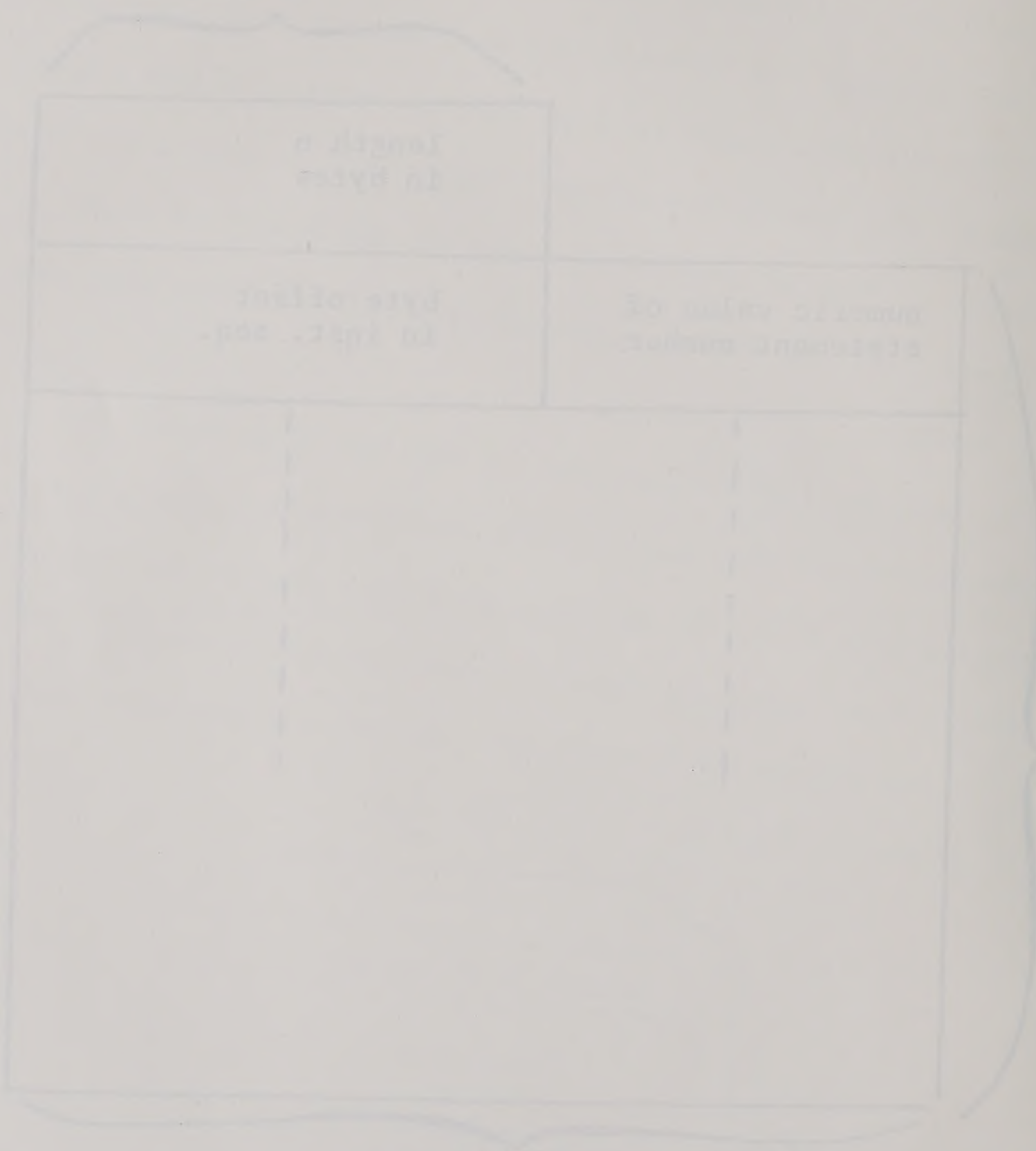
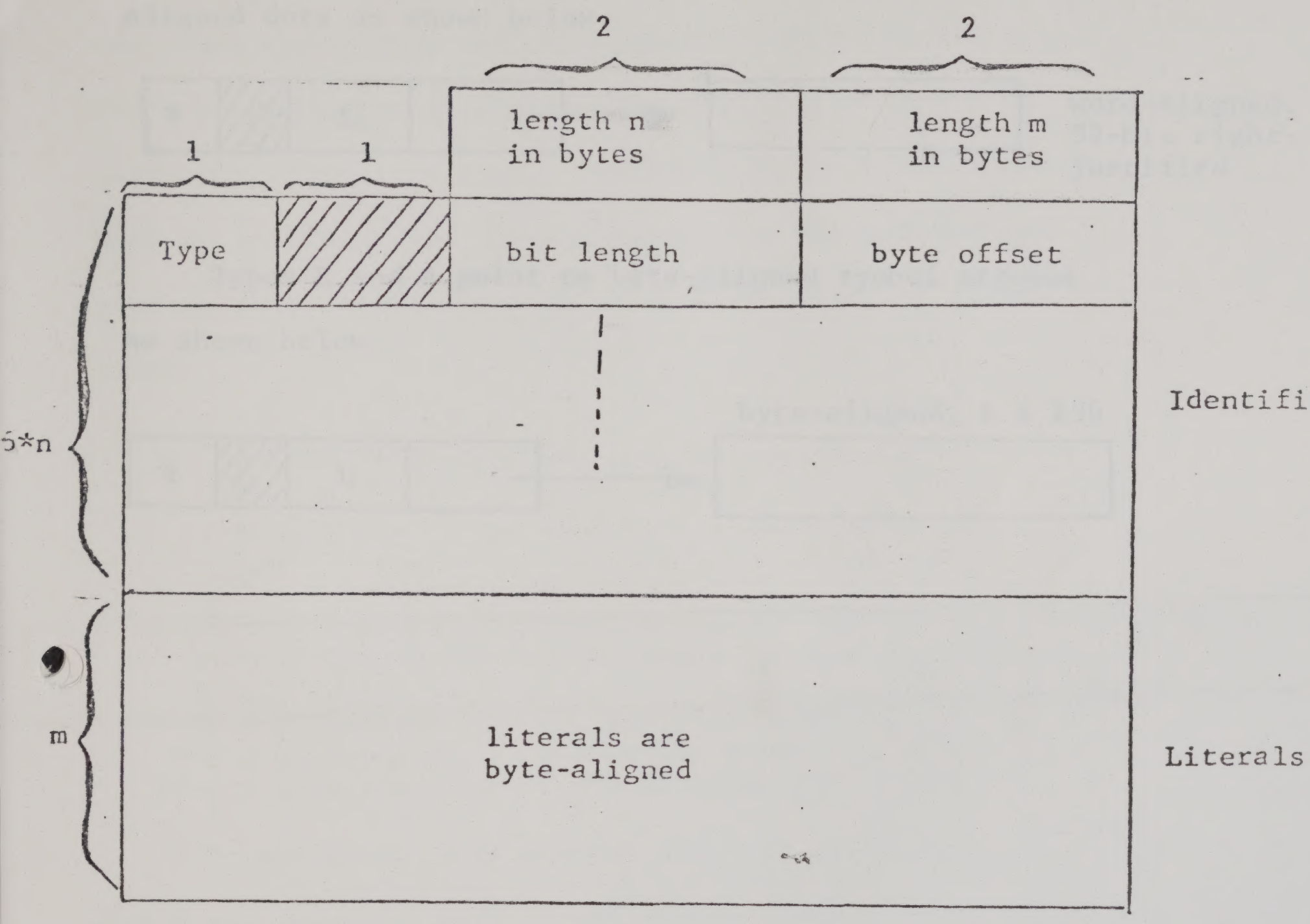


Fig. 4. Compiled Label Table



Literals and Identifiers are compiled as shown in Fig. 5.

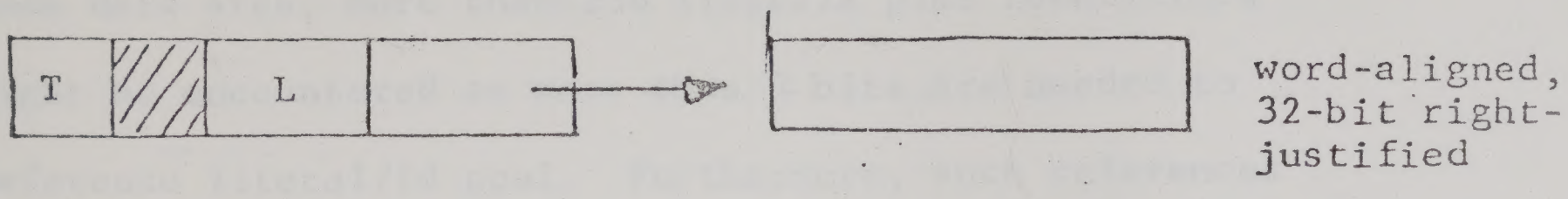


Legend:

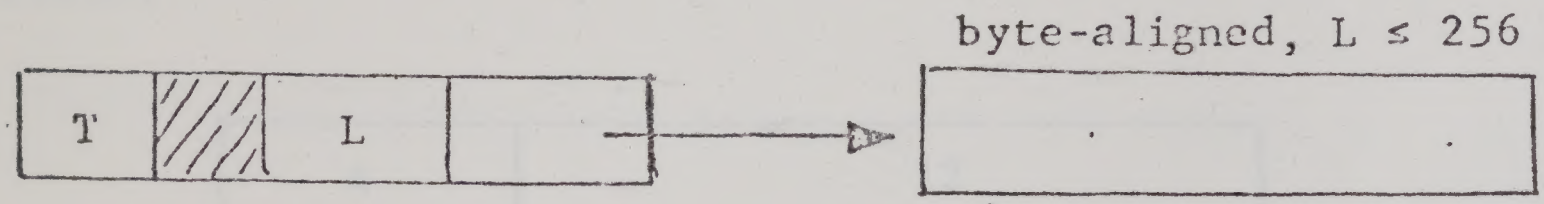
Type 0 = undefined
1 = B (binary)
2 = Ø (octal)
3 = X (hexadecimal)
4 = E (EBCDIC)
5 = A (ASCII)
6 = ED (EBCDIC encoded decimal)
7 = AD (ASCII encoded decimal)
8 = SB (signed binary, two's complement)

Fig. 5. Compiled Literals and Identifiers

Types B, Ø, X, AD, ED, and SB point to 32-bit word-aligned data as shown below.

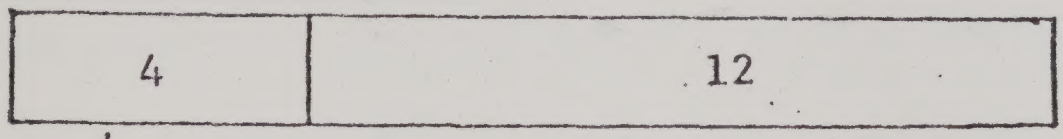


Types E and A point to byte-aligned symbol streams as shown below.



Instruction Format

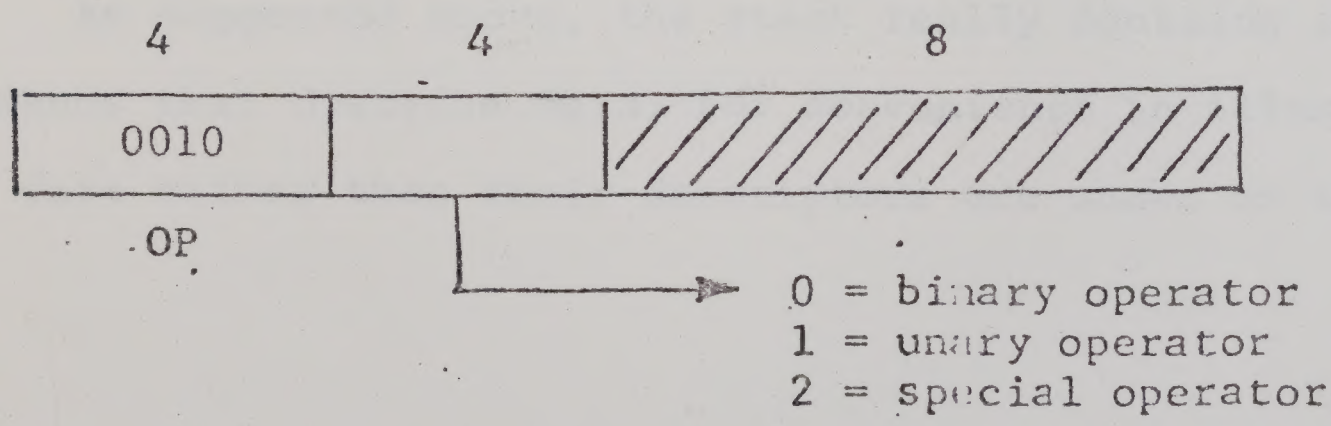
Since literals and identifiers will be stored in the same data area, more than 256 literals plus identifiers might be encountered so more than 8 bits are needed to reference literal/id pool. Furthermore, such references must be distinguished from operators in the instruction stream, so a 16-bit instruction will be used, as shown below.



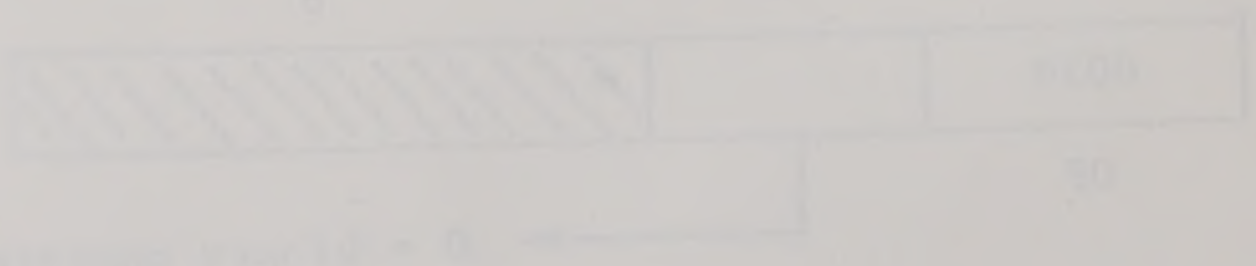
LD = 0 literal or identifier reference (12-bit positive integer)
IC = 1 12-bit two's complement integer constant
OP = 2 operator
AD = 3 address (12-bit positive integer)
ARB = 4 indefinite replication factor
NULL = 5 missing attribute of term

The operation code decoder picks up types 0, 1, 3, 4, and 5 and deposits them on top of the stack (TOS). LD is an index into the literal/identifier table, and AD is an index into the instruction sequence.

The decoder examines OP elements further:

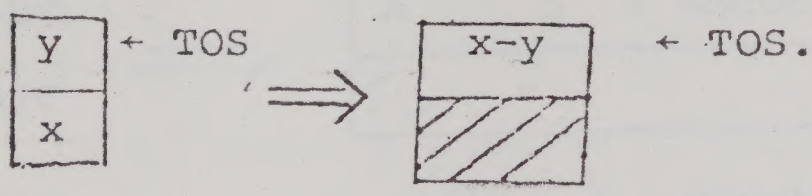


A
12

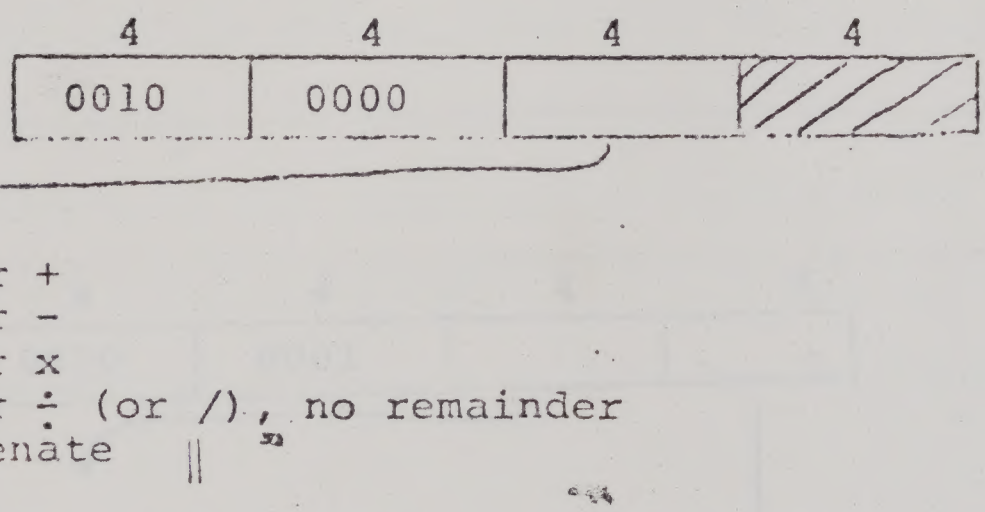


Binary Operators*

Let the TOS contain y and the next level, x. The binary operators compute x <bop> y, popping both x, y from stack, and put the result back on top of the stack.

e.g. x-y ⇒ xy-⇒ 

Binary Operator Encoding

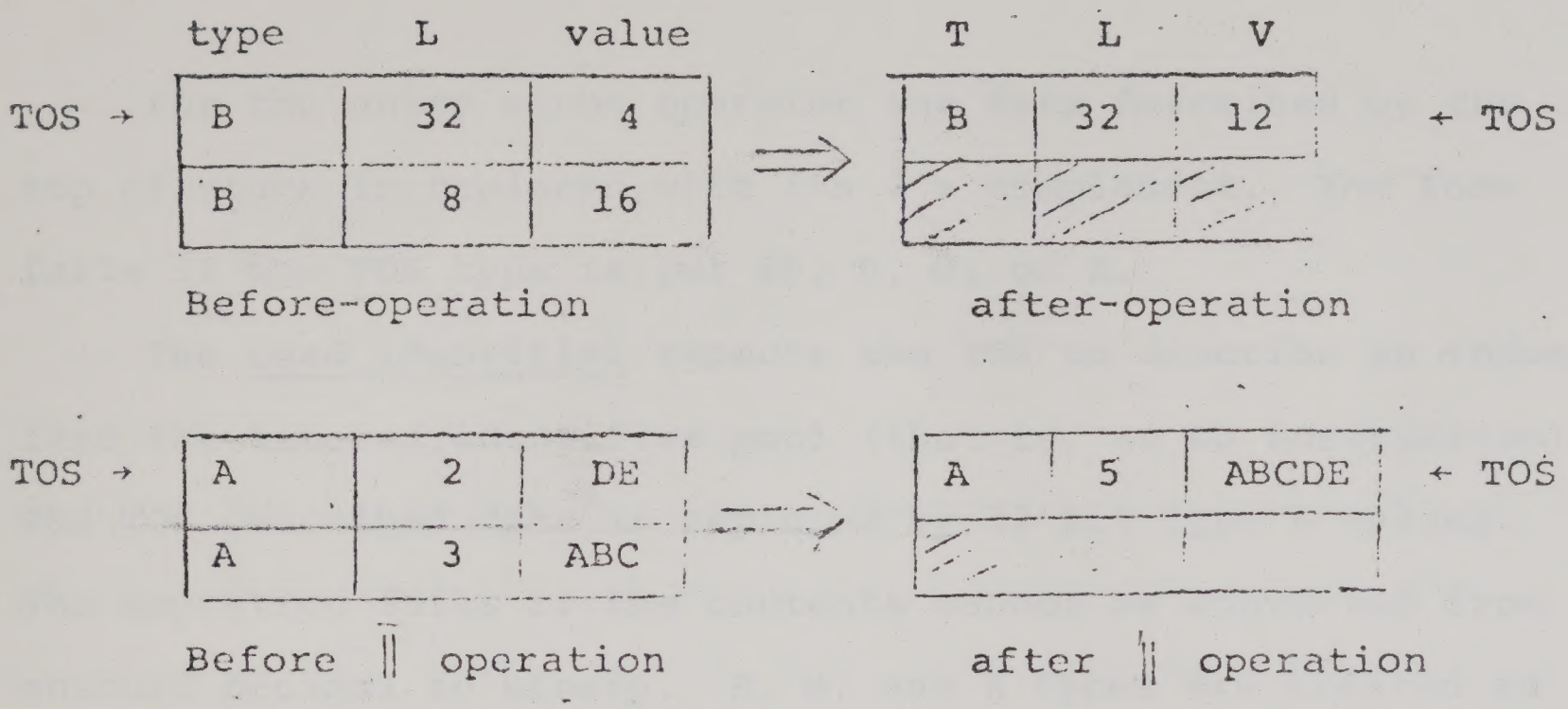


0 = integer +
1 = integer -
2 = integer x
3 = integer ÷ (or /), no remainder
4 = concatenate ||

All binary operators except concatenate expect the top two elements on the stack to describe type B, Ø, X, or SB. The result is always a 32-bit type B element. The concatenate operator fails unless both types are identical. For example:

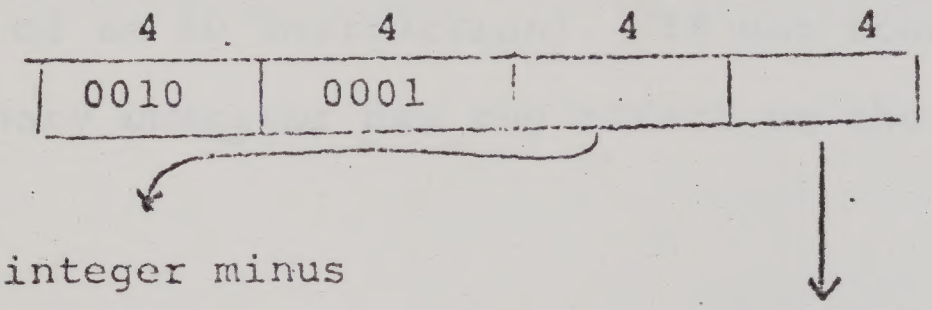
---

*As suggested above, the stack really contains instruction operands that describe data; for convenience in illustrations the data rather than their descriptors are shown on the stack.



No binary operator has any effect on the TRUE/FALSE flag.

Unary Operators



0 = integer minus

1 = load identifier

0 = evaluated contents (after dec → binary conversion)

1 = length field

2 = type field

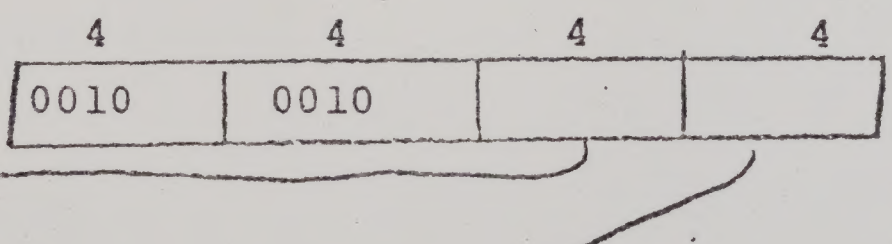
2 = Label Table Reference

For the unary minus operator the data described by the top of stack is replaced with its 2's complement. The form fails if the TOS type is not SB, B, Ø, or X.

The Load identifier expects the TOS to describe an index into the literal/identifier pool (that is, an LD instruction). The TOS described data is replaced by 32-bit type B values. The operation fails if the contents cannot be converted from encoded decimal to binary. B, Ø, and X types are treated as unsigned integers; SB is treated as 2's complement.

The Label Table Reference operator expects a 32-bit type B value described by TOS and searches for this label in the Label Table. If found, the TOS described data is replaced by the relative address in the instruction sequence of the label (in the form of an AD instruction). If not found, the form fails. No Unary operator has any effect on the TRUE/FALSE flag.

Special Operators



0 = store TOS.
1 = return
2 = branch — 0 = true, 1 = false, 2 = unconditional

3 = compare — 0 = .EQ. 2 = .LE. 4 = .GE.
1 = .NE. 3 = .LT. 5 = .GT.

4 = move input ptr — 0 = store current into initial
1 = store initial into current

5 = input call

6 = output call

0 = no compare
1 = compare

Store TOS

The TOS describes an index into the ID table and the next lower element in the stack describes a value to be stored. After execution, both elements are popped off the stack.

Return

The TOS describes a value to be returned to the routine which initiated the FORM MACHINE. The actual mechanism will be implementation dependent, but the FORM MACHINE will relinquish control after this instruction completes execution.

Branch

The TOS describes an index into the instruction sequence to be used as the new instruction counter (IC) if the branch conditions are satisfied. The branch instruction checks the state of the TRUE/FALSE flag register and either increments the IC by 1 or replaces it with the TOS described element. In any case, the TOS is popped.

Compare

The compare operator takes the two elements described by the two top stack entries and compares them (.EQ.,LT.,etc.). If n is at the top of the stack, and m is just below, then m.xx.n is performed, and the TRUE/FALSE flag is set accordingly. For .xx. = .EQ. or .NE. we must have identical type, length, and content for equality to hold.

The other boolean comparators will not be attempted if types are different (i.e., form fails), but for same types, B, Ø, X cause binary right-justified compares, and A, E, AD, ED cause left-justified string compares with the shorter string padded with blanks.

Move Input Pointer

This operator (no operands) replaces the Current Input Pointer with the Initial Input Pointer (back-up), or the

Initial Input Pointer with the current one (entry to rule).

Input Call

This is the most complex operator thus far encountered. It requires four operands from the run-time stack:

TOS	binary or null	length to find
	LD to literal or null	value (literal)
	binary code	input data type
	binary, arbitrary, or null	replication count

The input call operator can be invoked with the "no compare" flag set, in which case the value expression parameter is ignored and only the input type and length expressions are used. In this case, the input routine tries to find in the input stream as many characters of the required type (bits, digits, etc.) as needed to fill the length expression requirement. If successful, the TRUE/FALSE flag is set TRUE, the stack is popped to remove the input parameters, and the string obtained is described by the TOS. If the input stream cannot be matched then the parameters are popped off the stack, and the TRUE/FALSE flag is set FALSE.

If the "compare" flag is set, the input stream must be searched for the value expression. However, we must take some

care here to be sure we know what to look for. There are several cases:

a) The length expression parameter is greater than the length of the value expression but the type of input desired is the same as the value expression type. For B, Ø and X types, right-justify value expression in length-expression field, sign bit is extended left if type BS. If type A, E, AD, or ED pad on the right with blanks.

b) Same as a) but length is too small. B, Ø, and X type strings are truncated on the left. A, E, AD and ED are truncated on the right.

c) The type of the value expression and the type parameter differ. This case is deferred for discussion and presently is considered an error causing form failure.

If the input string matches, then the TRUE/FALSE flag is set true, the parameters are popped from the stack, and the resulting string is described by the TOS. Otherwise, the FALSE flag is set and the parameters are popped.

When a successful match is found the input subroutine always advances the Current Input Pointer by the appropriate amount. Since we are dealing at the bit level this pointer must be maintained as a bit pointer!

Output Call

This routine utilizes the same parameters as the input call, but operates on the output stream. The TRUE/FALSE flag is not disturbed by this operator. As for input, there

are four parameters on top of the stack, the length expression value, the value expression value, the desired output type, and the replication expression value. When there is a mismatch between the output type and the value expression type, a conversion must take place. The value expression is transformed into the desired output type and fitted into the field length specified by the length expression.

Truncation and Padding Rules

a) Character → character (A,E,AD,ED → A,E,AD,ED) conversion is left-justified and truncated or padded with blanks on the right.

b) Character → numeric and numeric → character conversion is right-justified and truncated or padded on the left with zeros. Beware! Two's complement numbers may be bollixed by this.

c) Numeric → character conversion is right-justified and left padded with blanks or left-truncated.

As for the unary operators, a numeric bit-string is treated as unsigned, except SB which is treated as two's complement. Thus we have:

```
(1,ED,X"FF",3) = E'255'
(1,ED,X"100",3) = E'256'
but (1,ED,SB"100000000",4) = E'-256'
```

If the output routine is able to perform the desired action, it advances the Output Stream Pointer, and pops all parameters from the run-time stack.

V. INSTRUCTION SET

t/id ref	LD <num>		Literal or identifier reference → TOS
t const	IC <num>		small 2's comp. integer constant → TOS
dress.	AD <num>		Address → TOS
indefinite	ARB		indefinite replication factor
l parameter	NULL		missing term attribute
d	ADD	TOS = x,y	y + x → TOS
btract	SUB	TOS = x,y	y - x → TOS
ltiply	MUL	TOS = x,y	y * x → TOS
vide	DIV	TOS = x,y	y/x →TOS
ncatenate	CON	TOS = x,y	y // x → TOS
ary minus	UNIN	TOS = x	-x → TOS
ad id value	LIV	TOS = LD x	V(LD x) → TOS
ad id length	LIL	TOS = LD x	L(LD x) → TOS
ad id type	LIT	TOS = LD x	T(LD x) → TOS
ok up label	LVL	TOS = x	AD x → TOS
	STO	TOS = x,y	y → x
turn	RET	TOS = x	return to caller with x
anch true	BT	TOS = AD x	AD x → Instr. counter
anch false	BF	TOS = AD x	AD x → Instr. counter
anch	BU	TOS = AD x	AD x → Instr. counter
mpare equal	CEQ	TOS = x,y	(y.EQ.x) → TRUE/FALSE fla
mpare not equal	CNE	TOS = x,y	(y.NE.x) → T/FF
mpare ≤	CLE	TOS = x,y	(y.LE.x) → T/FF
mpare <	CLT	TOS = x,y	(y.LT.x) → T/FF
mpare ≥	CGE	TOS = x,y	(y.GE.x) → T/FF
mpare >	CGT	TOS = x,y	(y.GT.X) → T/FF
ll output	OUT	TOS = r,t,v,l	(r,t,v,l) → output
ll input	IN { INC = compare INN = no compare	TOS = r,t,v,l	(r,t,v,l) → TOS
rrent → initial	SCIP	CIP → IIP	(store current input ptr → initial IP)
itial → current	SICP	IIP → CIP	(store initial input ptr → CIP)

VI. EXAMPLE COMPILATION

FORM SOURCE	ADDR.	INSTR.		GENERATED POLISH INSTRUCTION SEQUENCE COMMENTS
(NUMB.<=.1);	0	SICP		RULE PRELUDE
	1	IC	1	
	2	LD	0	REFERENCE TO NUMB
	3	STO		STORE IN NUMB
	4	SCIP		RULE POSTLUDE
1 CC(,E,,1:FR(99)),	5	SICP		RULE PRELUDE
	6	NULL		NO REPLICATION EXPRESSION
	7	IC	4	TYPE EBCDIC
	8	NULL		NO VALUE EXPRESSION
	9	IC	1	LENGTH
	10	INN		INPUT CALL WITH NO COMPARE
	11	AD	15	
	12	BT		SKIP RETURN IF INN SUCCEEDS
	13	IC	99	RETURN CODE
	14	RET		RETURN TO CALLER IF FAILED
	15	LD	1	REFERENCE TO CC
	16	STO		STORE INPUT DATA IN CC
LINE(,E,,121:	17	NULL		NO REPLICATION EXPRESSION
FR(99)),	18	IC	4	TYPE IS EBCDIC
	19	NULL		NO VALUE EXPRESSION
	20	IC	121	LENGTH
	21	INN		INPUT WITH NO COMPARE
	22	AD	26	
	23	BT		SKIP RETURN IF OK
	24	IC	98	RETURN CODE
	25	RET		RETURN TO CALLER IF FAILED
	26	LD	2	REFERENCE TO LINE
	27	STO		STORE INPUT IN LINE
:CC,	28	SCIP		SUCCESSFUL INPUT
	29	NULL		NO REPLICATION FACTOR
	30	LD	1	REFERENCE TO CC
	31	LIT		TYPE OF CC
	32	LD	1	REFERENCE TO VALUE OF CC
	33	LD	1	CC AGAIN
	34	LIL		LENGTH OF CC
	35	OUT		OUTPUT CC
(,ED,NUMB,2),	36	NULL		NO REPLICATION
	37	IC	6	TYPE IS ED
	38	LD	0	REFERENCE TO VALUE OF NUMB
	39	IC	2	LENGTH OF OUTPUT FIELD
	40	OUT		OUTPUT NUMB AS EBCDIC DEC.
(,E,E".",1),	41	NULL		NO REPLICATION

```
                        42      IC    4     TYPE IS EBCDIC
                        43      LD    3     REFERENCE TO E"."
                        44      IC    1     LENGTH TO OUTPUT
                        45      OUT         OUTPUT THE PERIOD
(,E,LINE,117),          46      NULL        NO REPLICATION
                        47      IC    4     TYPE IS EBCDIC
                        48      LD    2     REFERENCE TO LINE
                        49      IC    117   LENGTH TO OUTPUT
                        50      OUT         PUT OUT CONTENTS OF LINE
(NUMB.<=.NUMB+1:        51      LD    0     REFERENCE TO NUMB
          U(1));        52      IC    1     AMOUNT TO ADD
                        53      ADD         ADD TO NUMB
                        54      LD    0     REFERENCE TO NUMB
                        55      STO         STORE BACK INTO NUMB
                        56      AD    5     PLACE TO GO
                        57      B           UNCONDITIONAL BRANCH BACK
```

LITERAL/IDENTIFIER TABLE

```
0       NUMB
1       CC
2       LINE
3       E"."
```

LABEL TABLE

LABEL	OFFSET
1	5